A Peer-to-peer Purchase and Rental Smart Contract-based Application (PuRSCA)

Rafati Niya, Sina ; Schüpfer, Florian ; Bocek, Thomas ; Stiller, Burkhard

Abstract: This work introduces the design and implementation of an Android-based Peer-to-peer Purchase and Rental Application termed PuRSCA, which leverages Smart Contracts (SC) and the Ethereum public blockchain (BC). As a Device-to-device (D2D) communication protocol, WiFi-Direct is chosen to enable the P2P data transmission between two parties. This work results in a cost-efficient, secure, SC-based, P2P, and Decentralized application (Dapp). Evaluations on performance of this Dapp is specified in terms of its D2D deployment, transaction costs, scalability, security, and privacy.

DOI: https://doi.org/10.1515/itit-2017-0036

Sina Rafati Niya*, Florian Schüpfer, Thomas Bocek, and Burkhard Stiller

A Peer-to-peer Purchase and Rental Smart Contract-based Application (PuRSCA)

https://doi.org/10.1515/itit-2017-0036
Received December 1, 2017; revised July 30, 2018; accepted August 9, 2018

Abstract: This work introduces the design and implementation of an Android-based Peer-to-peer Purchase and Rental Application termed PuRSCA, which leverages Smart Contracts (SC) and the Ethereum public blockchain (BC). As a Device-to-device (D2D) communication protocol, WiFi-Direct is chosen to enable the P2P data transmission between two parties. This work results in a cost-efficient, secure, SC-based, P2P, and Decentralized application (Dapp). Evaluations on performance of this Dapp is specified in terms of its D2D deployment, transaction costs, scalability, security, and privacy.

Keywords: Smart Contracts, Blockchains, Ethereum, Peer-to-peer Applications, Purchase and Rental Contract, Decentralized Mobile Applications

ACM CCS: Computer Systems Organization → Architectures → Distributed Architectures → Peer-to-peer Architectures → Blockchains

1 Introduction

In the last decade, Business-to-Consumer (B2C) platforms dominated e-commerce business. With the introduction of initial Consumer-to-Consumer (C2C) platforms, a new form of markets emerged where private persons can come together and exchange goods and services directly [20]. The phenomenon of the rapid growth of the C2C market in the last years is often referred to as the "Sharing Economy" [20] with uses cases such as car and home rental [1, 23].

As of today, concluding an electronic (online) purchase contract between two parties requires centralized platforms to mediate the interests of seller and buyer. These platforms have to store the description of purchased items and their price, and customers can interact with such a platform to buy any item such as in ebay [12]. Online purchasing also requires centralized platforms, like banks or credit card institutes, to enable and operate safe and valid payments. The problem that all these platforms share is that they rely on a Trusted Third Party (TTP), the platform owner, to operate the platform. This results in many disadvantages for consumers. They need to register at each platform separately and typically have to give away their private data to the platform owners. Customers often have to pay transaction fees [5]. A TTP also determines a single point of failure for the platform, since (a) can be shut down due to platform owner not wanting to comply with a central (legal) authority or (b) it can be attacked.

A solution to overcome is central requirement of a TTP in a contractual agreement between two or more parties is offered by Smart Contracts (SC). The concept of SC was first introduced by [26] as a digital protocol that facilitating an agreement process between different parties without any intermediary by enforcing certain predefined rules that embed contractual clauses, like property rights, directly into hard- and software to make it expensive for a party to breach the protocol.

[26] outlines a hypothetical digital security system for cars, where a cryptographic key represents the ownership of that car and the cryptographic key is only transferred, if the terms implemented within the protocol are fulfilled. In this example, the owner of a car can withdraw the key from a leaser, who does not pay the monthly rent. At that time, SCs were not practically achievable, since no digital infrastructure existed that allowed the secure execution of such protocols without a TTP. This changed with the introduction of the cryptocurrency Bitcoin [4] and its underlying Blockchain (BC) technology.

The Ethereum BC [14] was introduced with a fully decentralized architecture that provides the potential in solving problems of centralized applications without a TTP. Ethereum enables development of Decentralized applications (Dapp) using SCs. Several scripting languages are supported by Ethereum, such as Solidity, a contract-oriented high level language whose syntax is similar to JavaScript (JS). Various types of apps can be implemented

***Corresponding author: Sina Rafati Niya,** Communication Systems Group CSG, Department of Informatics IfI, University of Zürich, Binzmühlestrasse 14, CH-8050 Zürich, Switzerland, e-mail: rafati@ifi.uzh.ch
Florian Schüpfer, University of Zürich, Binzmühlestrasse 14, CH-8050 Zürich, Switzerland, e-mail: flo.schuepfer@bluewin.ch
Thomas Bocek, Burkhard Stiller, Communication Systems Group CSG, Department of Informatics IfI, University of Zürich, Binzmühlestrasse 14, CH-8050 Zürich, Switzerland, e-mails: bocek@ifi.uzh.ch, stiller@ifi.uzh.ch

in Solidity, like voting, crowd funding, blind auctions or multi signature wallets [15].

To tackle the problem of an intermediary as with traditional contracts, such as the need for a TTP and time consuming paperwork to reach a legally valid documentation (e. g., description of traded objects and identities), this work here proposes the design and implementation of a Dapp to replace the centralized TTP-based solutions for purchase and rental contracts. This Dapp is developed for Android-based mobile phones with a P2P architecture which employs the Ethereum BC as its decentralized backend architecture. It addresses major challenges of a Dapp such as choosing the best protocol for Device-to-device (D2D) communications, providing trust, privacy, and security, developing SC code to be as cost efficient as possible with a high degree of scalability and an ability of handling and managing transactions for setting up successful contracts.

A critical aspect in developing apps using SCs, is the cost for persisting transactions in the BC. To overcome PuRSCA high costs of storing data, this paper introduces lightweight contracts along with full contracts. Further more, data types in the SC developed are selected carefully regarding units to result in the least possible cost. Also, a quantitative analysis was carefully followed to calculate SC deployment estimates in advance with a high degree of accuracy [30].

PuRSCA is based on a generic solution designed to cover a great area of trading objects. Users of this system, will be able to use this application while both parties of a contract have the application installed on their mobile phones and connected to the Ethereum BC.

This paper is organized as follows: Section 2 reviews related work. While legal aspects of purchase contracts are described in Section 3 and legal validity of SCs are explained, too. A brief overview of the WiFi-Direct communication protocol is provided in Section 4. Major design decisions are presented in Section 5, and the implementation of the system, including SCs and D2D connections, are discussed in detail in Section 6. Section 7, presents cost evaluations of the implemented SCs including empirical results and the analysis of privacy and security. Finally, conclusions and future work are provided within Section 8.

2 Related work

An investigation of existing apps reveals that only a few, such as Bogner's app [5], TransActive Grid [27], and Digix [11], exploit BCs and SCs to develop mobile apps with the purpose of a flexible (in terms of information from both parties to be presented to each other and accessing further parties information by D2D communications), legally valid, and safe transaction execution (guaranteed purchasing and renting transactions) for the purchase and renting use case.

An application presented in [5] employs SCs and the Ethereum BC for the conclusion of rental contracts. In this app, the creator of a contract can register rental objects that are stored in a dictionary and referenced by their IDs. SCs store the description of the item together with its rental price and deposit. When one party wants to rent an object, he/she scans its QR (quick Response) code with the camera of her smart phone. In Bogner's solution [5] a central SC is used to manage existing items that can be only rented by focusing on the commercial renting of items from a store. A draw back of such a centralized SC is that for registering each object, the owner has to pay a considerable amount of money for transaction fees to add the object's information into the public Ethereum BC. Bogner's App does not consider the renters' privacy. However, the Dapp focuses on direct transactions between two parties and also purchase contracts and supports the exchange of personal information between participants adjustably.

TransActive Grid is a not fully decentralized P2P energy trading platform that uses Ethereum-based SCs to manage transactions between participants in a local electric power microgrid [27]. Every owner of a "photovoltaic" facility installs a smart meter that keeps track of the surpluses he/she makes. The smart meter then updates the available surplus for this participant on an SC on the Ethereum BC. Interested buyers that living in the same neighbourhood can then interact with the contract to buy energy credits.

The Brooklyn microgrid [6] is the first renewable energy startup that uses the TransActive Grid system to create a local energy market. This system is not yet fully decentralized because it relies on the payment platform Paypal to conduct the financial transactions. However, it is dedicated to the application of trading SCs, a specific use case which is not usable for replacing traditional trading contracts.

Digix is an asset tokenisation service built on the Ethereum BC. Digix is offering physical gold bullions on the BC, more specifically, digital tokens (DGX tokens) linked to real world physical assets. DGX tokens are created ("minted") through an Ethereum SC with each token representing 1 gram of gold. Each DGX token can be linked definitely to a Proof of Asset (PoA) card of a physical gold bar. The PoA card is stored within an SC on the BC and contains information like the time stamp of the card creation,

Table 1: Comparison of Related Apps.

	BC Technology Used	Application Use case	Payment method	Client Platform	Fully Decentralized
Bogner Dapp	Ethereum	P2P rental contracts	Ether	Mobile	No
Transactive Grid	Ethereum	P2P energy trading	Paypal	Desktop	No
Digix	Ethereum	Physical asset tokenization	Gold and other metals	Desktop	No
PuRSCA	Ethereum	P2P rental and purchase contracts	Ether	Mobile	Yes

the gold bar serial number, the purchase receipt, and the audit documentation [10].

The PoA asset card is created through the Asset Registration Process. This process accepts a gold contract from a user and creates the PoA card that is linked to the Ethereum address of the user. To convert the PoA asset card to DGX tokens, a user can use the Minter SC that will hold the PoA card and return 1 DGX token per gram of gold. With the Recaster SC a user can exchange its DGX tokens back into PoA cards. To redeem the PoA card back to the physical gold bar, the user can trigger the Redemption Process [10].

The main benefit of DGX tokens is that their value is more stable than the cryptocurrency Ether (ETH), because they are directly backed by gold or other physical assets. Another benefit of DGX compared to traditional digital gold certificates is that this system does not rely on a centralized database that holds the information stored in a PoA card. Further, the related app is prone to "Man In The Middle" (MITM) attacks because users use a desktop client to log in, instead of a Web form [10].

A comparative overview of related apps is listed in Table 1. In this table, PuRSCA (Decentralized Purchase and Rental Contract Application) is the solution presented in this article. All apps in this table are developed in the Ethereum BC. Considering the use case and currency, the Bogner app [5] which is a Dapp designed for rental contracts is the closest implementation to the solution proposed in this work. Considering the client platforms, Transaction Grid [27] and Digix [11] are developed for desktop machines, while Bogner and SC Dapps are specifically designed for Mobile phones and considering decentralization. PuRSCA is capable of being fully decentralized by employing Ethereum light clients on the smart phones.

3 Legal aspects of purchase and rental contracts

Since legal aspects of purchase and rental contracts and their requirements need to be met to provide to users legally bonding contracts, a purchase contract should not only be accepted by law, but also be enforceable by law. This poses the question on properties required a purchase contract must have to offer a legally valid basis.

Law does not give a definition of a "purchase contract" but describes duties of the involved parties resulting from the conclusion of a purchase contract: The seller has to hand out the purchase item and to transfer the property of it to the buyer and it obligates the buyer has to pay the purchase price to the seller [8]. He/she also must accept the item that is offered to him in the purchase contract. If nothing else is negotiated or commonly accepted, the delivery and reception of an item must take place immediately. Finally, the buyer has to review the purchase item after reception and has to notify the seller about possible defects [27].

Conclusion of a purchase contract needs general rules on contract conclusions which in this case we consider the swiss federal law defined in *OR 1 ff* [25] have to be applied. The most important rules are briefly described here:

- **Conformable will:** It is crucial that both parties express their will to conclude the contract. It is also mentioned that this declaration can be explicit or implicit. This means that the relationship between the parties may be created orally, in writing or by conduct. For example, when a party accepts a good or a service for payment, this is considered a declaration of intent by conduct [7]. However, an offer in an online-shop is usually not considered an offer in legal terms, but an offer to the customer to send a bid to the seller. This means that the offer is made, when a customer orders a good or a service. The seller has to accept this offer for a contract to be formed. He/she can accept this offer either explicitly or implicitly by sending the goods to the customer [22]. Alternatively, [7] says that the applicant of an offer is not legally bound to it, if he/she adds a disclaimer of warranty to the offer or if such a caveat lies in nature or in the circumstances of an offer [24].
- **Requirements for Validity:** The purchase contract has to fulfill specific formal requirements for special kinds of contracts (for example when buying real

Table 2: Overview of wireless technologies in D2D communications.

	Bluetooth	BLE 4.0	WiFi-Direct	NFC
IEEE Standard	802.15.1	802.15.1	802.11 (a, b, g, n)	ISO 18092
Frequency (GHz)	2.4	2.4	2.4 and 5.0	0.01356
Max. Data Rate (Mbps)	1–3 (24 with HS)	1	11 (b), 54(g), 600 (n)	0.106, 0.212 or 0.424
Security	128 bit SAFER+	128 bit AES, user defined on app layer	256 bits AES-CCMP	short range, user defined on app layer

estate). Furthermore, the purchase contract has to conform to the barriers of contractual freedom. This means that the content of the contract can be chosen freely unless it violates the law or it is an agreement against public policy [27].

– **Articles of Agreement:** For the conclusion of a contract the parties need to agree at least on the essential articles of the agreement (Essentialia Negotii). For a purchase contract, both parties must agree on the price and the purchase item for the contract to be valid [8].

3.1 Legal validity of SC-based applications

A crucial aspect about SC-based apps while comparing to regular contracts is whether these contracts are legally valid or they need to adopt to some specific rules. The nature of offers in SCs are very different from an offer in an online shop. In a conventional online shop, sellers do not have to accept an offer from customers, and buyers only have to pay the purchase price if sellers confirmed the purchase. In contrast, SCs allows only one buyer per offer and account of the buyer is immediately charged after clicking on the "Buy" button. Furthermore, the SC is locked after the payment of the buyer is confirmed and neither party can withdraw the money any more.

It can be argued that such an offer is binding by nature, because the SC does not allow for the withdrawal of a payment after an offer is accepted by the buyer. To make sure that an offer made by seller is binding, it is better to specify this explicitly in the general terms and conditions of the app. For example, general terms and conditions of "Ebay" also specify these conditions for offers made by their customers [13].

Articles of an agreement are fulfilled, if the price and the purchase item are defined properly. Since the SC requires the seller to specify the price in a common currency, the first condition is satisfied. Regarding the definition of the purchase item, the law does not specify in which form and how detailed this description must be. Therefore, it has been assumed that a written description of a purchase item and its properties together with one or more pictures of it are considered sufficient.

Regarding formal requirements, it can be stated that every SC is valid as long as its content does not require this contract to fulfill special formal properties. This means that for example contracts involving the purchase of real estate, patents, designs, or trade markets cannot be concluded with the SCs developed within PuRSCA.

Considering the above mentioned requirements, the implemented PuRSCA approach is legally valid, since the following conditions are fulfilled: (1) General terms and conditions explicitly state that an offer made by the seller is binding by law. (2) Content of the SC does not violate the law and is not an agreement against public policy. (3) The content of the SC does not have special formal requirements by law.

4 Device-to-device communication

In order to provide the means of sending and receiving the contract description, identity data, and images of objects, owners, and costumers, there are few protocols to choose such as, Near Field Communication (NFC), WiFi, WiFi-Direct, Bluetooth, and Bluetooth Low Energy (BLE).

Wireless technologies may differ in various dimensions such as operating frequency range, data rate, security standards, and ease of utilization. Intuitively, the best protocol selection depends on the requirements and use cases. In PuRSCA, the most important factors for selecting a D2D protocol can be classified as high data rate, security, and ease of use. A brief overview of D2D communication protocols are listed in Table 2 and a comprehensive comparison is done by authors in [31].

WiFi-Direct is an extension of the IEEE 802.11 protocol and it is developed for direct communication of two or more devices in the absence of a distribution network. Instead of building on top of the 802.11 ad-hoc mode, which

was never widely adopted and lacks efficient power saving support and extended QoS capabilities, it extends the infrastructure mode and allows the devices negotiate for the role of the Access Point. WiFi-Direct inherits all the enhanced QoS, power saving, and security mechanisms of the infrastructure mode [31].

WiFi-Direct uses the WiFi Protected Setup (WPS) procedure to secure the connection with minimal user intervention. WPS is based on the WiFi Protected Access 2 (WPA2) protocol. WPA2 implements the IEEE 802.11i standard and provides data confidentiality and integrity by using the Advanced Encryption Standard (AES)-CCMP cypher. When used in Personal mode, a Pre-Shared-Key (PSK) must be present both at the AP and the client for the mutual authentication. The 256 bit PSK is usually generated from a plain text pass phrase that must be entered on both devices. After the authentication, a set of temporary keys is exchanged between the AP and the client which are regularly updated [28]. WPA2 is considered secure against most attacks like man-in-the-middle (MITM), authentication forging, replay, key collision, weak keys, packet forging, and brute force attacks [3].

Considering throughput, since both the user profile and the contract can contain high resolution images, the payload can easily exceed 1 MByte. Both classic Bluetooth and BLE do not provide the required bandwidth for transmitting larger files in a reasonable time, especially since the achievable data rates will be below the theoretical limit. Bluetooth 3.0+HS could provide up to 24 Mbps, but it has not been widely used among smart phone users as well as NFC which usually users don't know how to run the NFC or they can't find the exact location of the NFC hardware in their cellphones. Therefore, WiFi-Direct is selected as it can be easily used and provides the required throughput.

Even though WiFi-Direct uses 256 bit key length for encryption, it is still vulnerable against Man-In-The-Middle (MITM) attacks during the WPS authentication procedure. While, both Bluetooth and BLE offer OOB and PIN authentication, to exchange keys [19, 21]. Considering related security issues of user profile, which contains personal information, the data transmission should be encrypted appropriately. Although the exchanged data is personal and allows a potential adversary to identify the parties, it does not provide her with information (i. e., user's Secret key (SK)) that could be used to steal money out of a contract or to sign a contract. Even if both parties could sign a contract with a certified signature in future, the SK would never be transmitted to the other device. Therefore, security is certainly important but it does not have the same priority as data throughput has in this use case.

5 Design

Based on the discussion in previous sections, some of the most important factors of designing PuRSCA can be specified as, automating the processes for data and money transactions, ensuring both parties with guaranteed transactions, identity verification, reliable decentralized data storage, trust to the Dapp and its underlying technologies, fast and secure P2P and D2D communications, and high user privacy.

PuRSCA is designed to operate and to be used as a BC-based system. Within this system, each two parties of a contract are able to setup contracts in a trustable P2P system, even though there is no direct trust between them. This feature derived by the public Ethereum BC nature.

General architecture PuRSCA is designed with two main utilization methods. In the first method, the users install Ethereum light client on their smart phones and thus, they are connected directly to the Ethereum BC as shown in the right side of the Figure 1. In the second method, users need to connect to an Ethereum client which may be installed on a their personal laptop, as shown in the left side of the Figure 1.

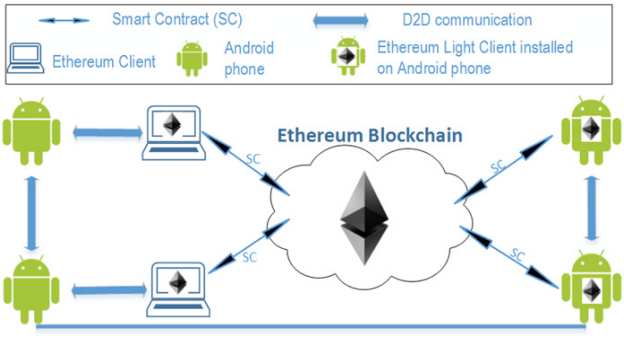

Figure 1: General System's Architecture – Users can connect to BC via full clients or via light clients.

Identity provision as one of the key aspects of this Dapp is designed to be possible in two methods. The first method is using QR code readers in which each party can read the other party's identity by reading his/her QR code image. Second method is to send and receive identity data via a D2D connection.

General design of this system is meant to provide flexibility in choosing ID information to be sent to the other side of contract. Being flexible is determined here as what users experience while setting up contracts. In PuRSCA, users are not forced to provide any documents and they can negotiate with each other (here bilaterally only, as

discussed above) and send to each other, e. g., ID information off-chain. With such an option of flexibly adapting to current contractual needs, users are able to insert the least possible amount of their ID information. Even though users have already entered all those information (e. g., 1: QR code and 2: image) in their profile, they can simply choose what to send and what to preserve from being sent.

Another important design decision in this system is to establish trust for both sides of contracts. In PuRSCA, parties of a contract have to deposit the same amount of money in terms of *Ether* to avoid any malicious behaviour.

5.1 System utilization

To use PuRSCA, users have to install the application and obtain a Public Key (PK) and Secret Key (SK) from the Ethereum BC. Sequence diagram of purchasing process is shown in Figure 2. In the trading process, the seller creates contracts and has to pay a deposit. Next, he/she sends the contract information to buyer and provides his/her ID information consecutively. The buyer has to send his/her ID to seller in return and execute the buying function. After the sold object is delivered to the buyer, he/she confirms reception of that object. In the next step, the deposited amount will be sent back to both parties and the seller receives the price for the sold object.

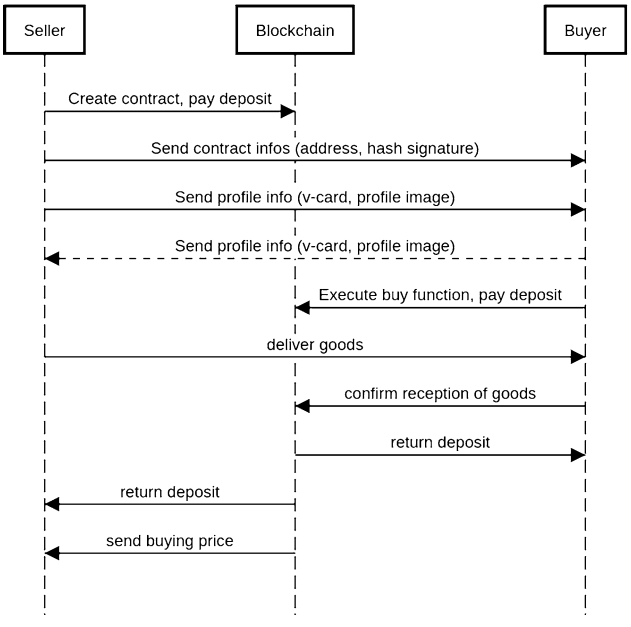

Figure 2: Designed Purchase Sequence Diagram In The Proposed System.

In the renting use case, PuRSCA designed to ask the hierer to execute the reclaim function after he/she received back the rented object from renter (cf. Figure 3). In the next step, renting price is calculated by the rental-SC and renter executes the return function to pay the rent price. Finally the deposited amount by two parties will be sent back to each of them (cf. Figure 4).

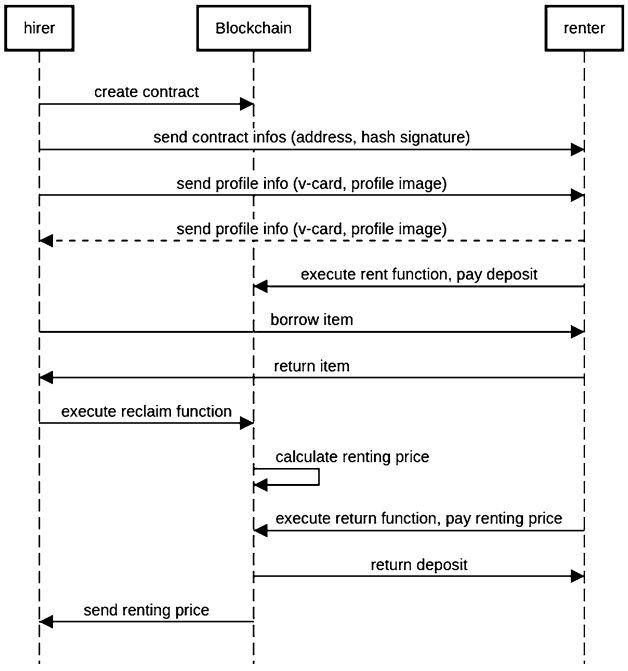

Figure 3: Designed Renting Sequence Diagram In The Proposed System.

6 Implementation of the P2P solution

Implementation of PuRSCA includes developing the purchase and rental SCs, Android application, D2D communications, installing Ethereum clients, and eventually connecting all parts together.

6.1 Rental and purchase smart contracts

Purchase and rental contracts are based on the "Simple Trading Contract" presented in Solidity official documentations [15]. Users can setup purchase or rental contracts with the Android client by creating new contracts or, using already existing ones. In the purchase SC, to conduct a safe purchase both seller and buyer have to pay a deposit before transaction can take place and deposited as-

sets are only released when the buyer confirms reception of the item.

In the rental SC, on one side hirer can specify renting and deposit fees for an item. On the other side, renter has to pay the deposit beforehand and only gets it back after he/she returned the item and paid the renting fee.

In each contract, description of an item is composed of a title, textual description, and an array of image signatures. An image signature is calculated by using the SHA256 cryptographic hash function of the images taken by the Android client. The reason for not storing a whole image in the Ethereum BC is the cost associated with the high storage usage. Assuming that a PNG or JPG image with reasonable resolution has a size of 300 KByte, and cost of storing one 256 bit word on the Ethereum BC is 20,000 units of *gas*, cost for storing one image can be calculated by:

$$P_{storage} = 937.5 \times 20000 \times P_{gas} \times P_{wei} \quad (1)$$

Where *Pstorage* is the price for storing the image in USD, *Pgas* is the gas price and *Pwei* is the dollar exchange rate for 1 *wei*. Absolute price can vary because both gas price and exchange rate for *Ether* have a high volatility. As of July 16, 2017, the average gas price is 4 *gwei* [15] and the dollar exchange rate for 1 *Ether* is approximately 175 $ [8]. This would result in a price of 131.25 $ for storing one single image!

Every contract also stores *address* of the 2 contractual partners as well as *price* and *deposit* for the item. The *deposit* is used to ensure contractual compliance of the parties. Finally, contract stores a boolean value that indicates whether the parties should exchange personal information (Listing 1.1 line 2). This indicator is used by client app to determine whether personal information like addresses, photos or other data should be exchanged between the parties. Every contract can be aborted in certain circumstances and therefore declares the aborted function and an aborted event (Listing 1.1 lines 7).

6.1.1 Purchase smart contract

Purchase contract is derived from the *TradeContract* and it is a modified version of the purchase contract presented in the official Solidity documentation [15] as shown in Listing 1.1.

Listing 1.1: The Purchase Contract

```
1  contract Purchase is TradeContract{
2  function Purchase(string _title, string
     _description, bool _verify, bytes32[]
     _imageSignatures) payable
3  TradeContract(_title, _description, _verify,
     msg.value / 2, msg.value / 2,
     _imageSignatures){
4    if (2 * price != msg.value) throw;}
5  event purchaseConfirmed();
6  event itemReceived();
7  function abort()
8  onlySeller
9  inState(State.Created){
10 aborted();
11 state = State.Inactive;
12 if (!seller.send(this.balance)) throw;}
13 function confirmPurchase()
14 inState(State.Created)
15 require(msg.value == 2 * price)
16 payable{
17 purchaseConfirmed();
18 buyer = msg.sender;
19 state = State.Locked;}
20 function confirmReceived()
21 onlyBuyer
22 inState(State.Locked){
23 itemReceived();
24 state = State.Inactive;
25 if (!buyer.send(deposit) || !seller.send(
     this.balance)) throw;}}
```

To give an incentive to a seller to not betray a buyer, the seller has to transmit Ether in the constructor of the contract. As can be seen in (Listing 1.1 lines 3–5), the value provided must be dividable by 2, otherwise the SC throws an exception. Thus the actual price of the item is only half the value provided in the constructor (Listing 1.1 line 15). The deposit is refunded to the buyer and the rest of the balance stored in the contract is sent back to the seller. Notice that the state is changed before the Ether is transmitted with the "send" function (Listing 1.1 lines 11–12). This prevents an attacker from calling the function again from its fallback function.

When the buyer executes the "confirmPurchase" function (cf. Listing 1.1 line 17), he/she also has to pay a deposit together with the actual purchase price. The state of the contract is then set to "Locked" (cf. Listing 1.1 line 19). From this point on, the funds of the 2 parties are locked and can only be released when the buyer confirms that he/she received the item (cf. Listing 1.1 line 22). Locking of deposits makes sure that no party has a monetary advantage in betraying other side since both have invested the same amount of money in the contract and non of them can get their deposit back when the other party does not comply.

As long as the contract is in the state "Created" meaning that buyer has not accepted the offer, seller can execute the "abort" function and the balance stored in the contract is refunded to him. In the "confirmReceived" function the buyer confirms that he/she received the item. The state of the contract is set to "Inactive" meaning that no further interaction with it is possible (cf. Listing 1.1 line 24).

Every time the state of the contract changes, an event is emitted on the event log. This allows the client app to update its UI and inform the buyer or seller that the state has changed.

6.1.2 Rental contract

In the rental contract as presented in Listing 1.2, buyer replaced by renter and the seller assumed as the role of a hirer. The hirer defines details of the rent item together with its renting price and deposit in constructor of the contract (Listing 1.2 line 5). Renting price is provided in *wei* per second and the total renting time is calculated (cf. Listing 1.2 line 28). As long as the contract is in the "Created state", hirer can execute the abort function (cf. Listing 1.2 line 12). When the renter executes the rent function (cf. Listing 1.2 line 4), he/she has to pay the deposit for the item. State of the contract is set to Locked and start time for the contract is initialized. When the renter wants to return an item, the hirer executes the *reclaimItem* function (Listing 1.2 line 41) which calculates the renting fee based on the renting time and the price (Listing 1.2 lines 22–29) and sets the state of the contract to *AwaitPayment* (Listing 1.2 line 31). In the *AwaitPayment* state, renter can execute *returnItem* function, which then returns the deposit and the change to the buyer. The renting price is sent to the renter. In the *AwaitPayment* state, the hirer can also execute the *reclaimItem* function again in the case the renter does not pay in time.

Listing 1.2: The Rental Contract

```
1  contract Renting is TradeContract{
2  uint256 private rentingFee;
3  uint256 private start;
4  function Renting(string _title, string
    _description, bool _verify, bytes32[]
    _imageSignatures, uint _deposit, uint
    _rentingPrice)
5  TradeContract(_title, _description, _verify,
     _deposit, _rentingPrice, _imageSignatures)
    {}
6  event itemRented();
7  event itemReturned();
8  event paymentRequested();
9  function abort()
10  onlySeller
11  inState(State.Created){
12    aborted();
13    state = State.Inactive;}
14  function rentItem()
15  inState(State.Created)
16  require(msg.value == deposit)
17  payable{
18    start = now;
19    itemRented();
20    buyer = msg.sender;
21    state = State.Locked;}
22  function calculateRentingFee()
23  returns(uint256){
24    if(state == State.Inactive)
25        return rentingFee;
26    if(start == 0)
27        return 0;
28  uint256 rentingTime = (now - start);
29  return price * rentingTime;}
30  function returnItem()
31  inState(State.AwaitPayment)
32  onlyBuyer
33  require(msg.value >= rentingFee)
34  payable{
35    itemReturned();
36    state = State.Inactive;
37    uint change = rentingFee - msg.value;
38    if (!buyer.send(deposit + change)||!
    seller.send(this.balance)) throw;}
40  function again
41  function reclaimItem()
42  onlySeller
43  payable
44  require(state == State.Locked || state ==
    State.AwaitPayment){
45    rentingFee = calculateRentingFee();
46    paymentRequested();
47    state = State.AwaitPayment;}}
```

6.2 Android application

Some of the most important pages of the PuRSCA are shown in Figures 4 and 5. Figure 4-a illustrates a contract created with the information of a Laptop to be sold. As shown in this image, picture and description of the laptop are available. Figure 4-b, illustrates the interface with which contract information will be sent to the buyer using WiFi-Direct. In the Fig. 4-c, buyers profile is illustrated before sending the contract info to him, the seller could read these information by reading the QR code of the buyer's profile. Figure 4-d, represents a contract's information. In this part, buyer/seller could abort the contract process.

As illustrated in the Figure 5-a, users are able to access their old contracts by choosing the *Add existing contract* instead of creating a new contract. Figure 5-b shows a list of previous contracts made by one client.

In PuRSCA, as a P2P system, Android clients use service interfaces of the Java library to load and deploy SC. The Java Web3j library [29] is used to wrap the interface of the SCs in Java classes which is then used by the Android client to execute transactions on the BC. Web3j uses the JSON-RPC interface of an external Ethereum client that is connected to the Ethereum network to deploy contracts and interact with them as shown in Figure 6. All SCs in the proposed Dapp are developed with Solidity language and their compiled byte-code is stored in the associated Java wrapper class.

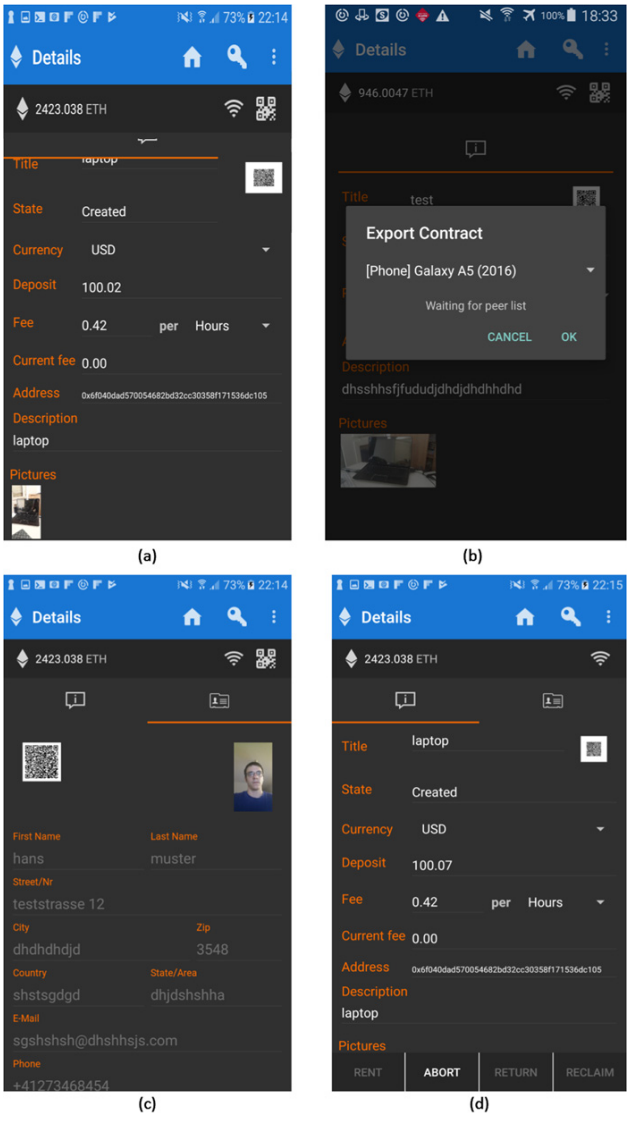

Figure 4: Process of purchasing a contract, (a) Information of a contract before setting up, (b) Setting up a contract, (c) Buyer identity information, and (d) Created contract's information.

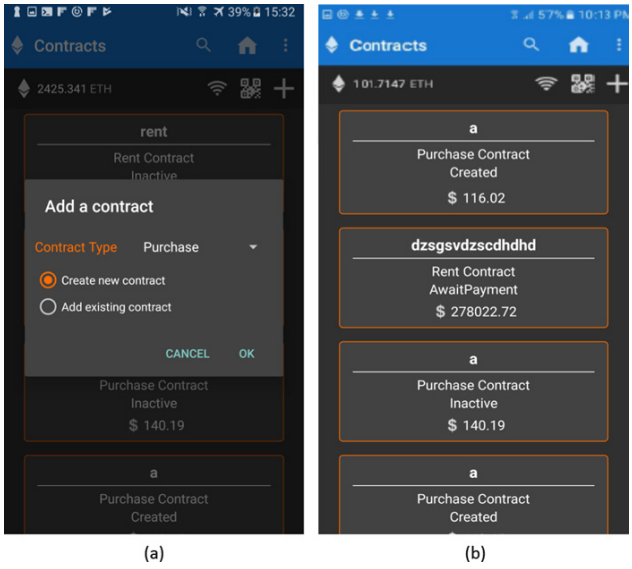

Figure 5: (a) UI for Setting Up New Contracts, (b) Accessing Previous Contracts.

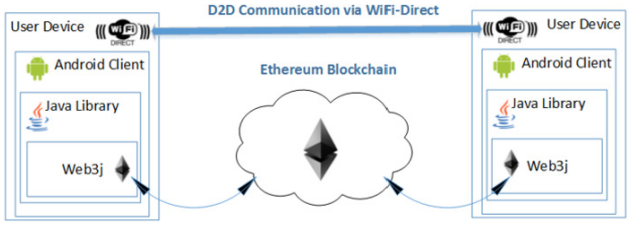

Figure 6: Implementation Architecture Integrating Android and Ethereum Clients.

6.3 Ethereum client and SC integration

A full Ethereum client on a remote machine is used to integrate the Android client with the BC. There are several Ethereum client implementations available. Go-Ethereum [18], the Google's Go implementation called Go-Ethereum (Geth) is used in this Dapp.

To communicate with the Android client form another process on the same or on another machine, the JSON-RPC protocol is used. Web3j Java library [29] is used to deploy SC on the BC and to interact with SC from the Android client. Web3j implements the JSON-RPC interface of the Ethereum client and provides high-level access to a transaction and its content.

7 Evaluations

It is economically important to develop SC-based apps with the least possible deploying and transaction costs. For that purpose, first step is to being aware of which parameters of developed SC affect the total cost of deploying contracts and sending transactions.

7.1 Costs

In this section, empirical results of deploying SCs are compared to the estimated costs, which are calculated with introduced cost estimation functions based on the experiences with SCs during the past 3–4 years [32]. Result of these comparisons indicate the high accuracy of provided estimation functions. One general outcome of these evaluations is the difference between light and full contracts which obviously light version has less deployment costs and this explains and proves the incentives of developing light SC.

Table 3: Measured Deployment Costs.

cost in gas and in (USD)	Empty Contract	With One Image (32 Byte)	With Two Images (64 Byte)	With Three Images (96 Byte)
Purchase Contract	691,023 (3.05)	697,617 (3.08)	719,104 (3.17)	740,591 (3.27)
Rental Contract	760,550 (3.35)	797,144 (3.52)	818,631 (3.61)	840118 (3.70)
Purchase Contract-light	389,702 (1.72)	389,702 (1.72)	389,702 (1.72)	389,702 (1.72)
Rental Contract-light	486,946 (2.15)	486,946 (2.15)	486,946 (2.15)	486,946 (2.15)

Table 4: Estimated Deployment Costs.

cost in gas and in (USD)	Empty Contract	With One Image (32 Byte)	With Two Images (64 Byte)	With Three Images (96 Byte)
Purchase Contract	644,800 (2.84)	664,800 (2.93)	684,800 (3.02)	704,800 (3.11)
Rental Contract	737,000 (3.25)	757,000 (3.33)	777,000 (3.43)	797,000 (3.52)
Purchase Contract-light	363,600 (1.60)	363,600 (1.60)	363,600 (1.60)	363,600 (1.60)
Rental Contract-light	454,000 (2.00)	454,000 (2.00)	454,000 (2.00)	454,000 (2.00)

The actual deployment costs in *gas* for the rental and purchase SC were evaluated using the browser Solidity online compiler [15] and the local Ethereum test client. For the median *gas* price, a value of 21 *gwei (gigawei)* was assumed [17]. For the USD exchange rate, the average value of 210 dollar per "Ether" since May 2017 was taken as a reference value [16].

According to [15], main costs for deploying a contract are the costs associated with storing the contract code ("CREATEDATA") with cost of 200 *gas* per byte and the costs for storing additional data on the storage of the contract ("STORAGEADD") with 20,000 *gas* per 256 bit word. Further, in addition to the 21,000 *gas* for a normal contract transaction, 32,000 *gas* have to be paid for a transaction that creates a contract [10] concluding in a cost of 21,000+32,000=53,000 *gas*.

Deployment costs for a contract depend on the size of the contract code and the amount of bytes that it assigns to the storage in its constructor. Since both the purchase and the rental contract do not perform any calculation intensive work in their constructors, their deployment costs in "Gas" can be estimated using the following formula:

$$C_{gas} = (53000 + 200 \times N_{bytes} + 20000 \times N_{words}) \quad (2)$$

Where C_{gas} is the total transaction cost in gas, N_{bytes} is the contract size in bytes and N_{words} is the number of 256 bit words that are initialized in the constructor. To calculate the deployment price in USD, the gas usage has to be multiplied with the *gas* price P_{gas} and the USD exchange rate for the *Ether* $P_{exchange}$:

$$C_{dollar} = C_{gas} \times P_{gas} \times P_{exchange} 10^{-18} \quad (3)$$

Table 3 summarizes the measured costs for deploying the rental and purchase contracts in full and light deployment mode using different amounts of additional data (images) to store. As it is shown in this Table, in all the three tested SCs with different number of images as additional data to be stored, costs for light contracts are significantly lower than full contracts as their source code use less space. Further, they use less storage by storing all content attributes (text and images) in a single 32 Byte hash. Therefore, deployment costs are constant and independent of the amount of additional data sent in constructor. Measured values are close to the values estimated by the simplified Equations 2 and 3 and are shown in Table 4. The Mean Absolute Percent Error (MAPE) is 5.8 %.

7.1.1 Transaction costs

Transaction costs for the purchase and rental contracts are dominated by the fix transaction costs "GTX" of 21,000 *gas*, the costs for adding a 256 bit word to the storage "STORAGEADD" of 20,000 *gas*, the costs for modifying a word on the storage "STORAGEMOD" of 5,000 *gas* and the costs for making a call from the contract that contains *Ether* "GCALLVALUETRANSFER" of 9,000 *gas*. Other costs are not significant since no major computation is done in any of the transaction functions. Therefore, to estimate the transaction costs, the formula 4 is used which yields to very good results with the MAPE of only 3.8 %.

$$C_{gas} = 21000 + 20000 \times N_{words\_add} + 5000 \times N_{words\_mod} + 9000 \times N_{tx} \quad (4)$$

Table 5: Real and Estimated Transaction Costs.

cost in gas and in (USD)	Abort	Confirm Purchase	Confirm Received	Rent Item	Reclaim Item	Return Item
Measured Costs	49,584 (0.22)	62,890 (0.28)	41,657 (0.18)	82,721 (0.36)	41,913 (0.18)	48,146 (0.21)
Estimated Costs	50,000 (0.22)	61,000 (0.27)	44,000 (0.19)	81,000 (0.36)	44,000 (0.19)	45,000 (0.20)

Where C_{gas} is the total cost in gas, N_{words_add} is the number of 256 bit words added to the storage in the transaction, N_{words_mod} is the number of words that are modified in the transaction and N_{tx} is the number of messages with *Ether* that are sent in the transaction function (e. g., for refunding *Ether* to the buyer or seller). The total costs in USD can be estimated by Equation 3.

Table 4 compares the measured transaction costs for all transactions of the rent and purchase contracts to the estimated costs with Equation 4. It was found that a variable is only initialized on the storage when a value different from 0 is assigned to it. For example, the *state* variable of the contract is only added in the *abort* or *confirmPurchase* function, when the value changes from the initial state. This has to be taken into account when using the formula.

7.2 Privacy

This section discusses how privacy issues that arise from storing and exchanging personal user data are addressed in the Dapp. It further discusses possible privacy issues that can rise when storing contract details on the BC and how they can be avoided.

Privacy of personal user data

In the proposed Dapp, all the profile information including profile images, are stored on internal storage of Android app to prevent other apps on the device access this data. However, it does not provide protection against an attacker that has physical access to a non-encrypted file system or against an attacker that has root access on the operating system [2].

Privacy of contracts

Storing contract details in plain text on a SC can cause privacy issues because the addresses of the seller and the buyer are also stored on the contract and are therefore publicly accessible. A party that knows a person with a particular address (e. g., when it exchanged contract or user data with that person in the past) can look up details of other contracts that were signed by that person. To prevent that, the light deployment option can be used. When using light deployment, only the hash of the details that not meant to be stored on the BC will be stored in the SC.

7.3 Security

In PuRSCA, accounts can either be managed locally on the Android device or remotely on the server running the Ethereum client. When using the *WalletAccountService* the local wallet file is decrypted using the password provided by the user and the credentials are used by the *RawTransactionManager* to sign every transaction with the SK belonging to that account. *WalletAccountService* provides protection against eavesdropping since the wallet password is never sent to the Ethereum client. It also provides protection against MITM attacks since a transaction cannot be altered any more after it has been signed with the SK. Password for a wallet file is never persisted on the file system and the SK of an unlocked wallet file is only stored in volatile memory.

There are only two scenarios left in which an attacker could obtain or use the SK of an account:

- The attacker has root access on the operating system and can intercept the password from the user by using a key logger.
- The attacker has physical access to the phone and the account is still unlocked. In this case the attacker could use the account to sign rent or purchase contracts and transmit money to her own account.

Regarding the D2D communication security, as mentioned in section 4, when user data is exchanged over WiFi-Direct it is always encrypted using WiFi Protected Setup (WPS). WPS uses WPA2 to encrypt and authenticate data and offers good protection against eavesdropping and brute-force attacks but it is vulnerable against MITM attacks during the short time frame in which the encryption keys are exchanged [9].

8 Conclusions

Traditionally, setting up purchase or rental contracts between two parties requires the provisioning of complete

and accurate identity information of both parties besides the details of exchanged objects. Hitherto proposals and apps have followed a centralized approach to ease the steps required to be taken. However, these apps maintain the disadvantage of centralized architectures. Most important challenges in designing such apps are the P2P communications between parties for transferring contract information, providing identity information while ensuring user privacy, legal binding, creating trust in the system, and data security.

To overcome paper-based preparations for setting up legally valid purchase and rental contracts between two parties and to solve contracts handling in centralized apps, this paper proposed the Peer-to-Peer Purchase and Rental Smart Contract-based Application (PuRSCA). PuRSCA's goal of satisfying functional and legal requirements of an automated purchase and rental contracts adopted the Ethereum BC. The P2P architecture of this Dapp enables highly viable data rates in D2D communications by using WiFi-Direct. In addition to the design of a full contract, the Dapp introduced light-weight contracts, which enable the opportunity of transferring identity information between users without the need to store them in the public BC and, therefore, reduces costs.

Furthermore, trust in PuRSCA is provided by designing the deposit-based SC to guarantee seller and buyers loyalty to contract and money transfer. High privacy provided by design as this Dapp is completely independent of any TTP and any seller can specify contract details including price, deposit, textual description, and images to deploy a contract. Contract details can be exchanged either by scanning a QR-code of the contract or by using WiFi-Direct. In the latter case, parties can flexibly decide which personal information they want to share. Contracts are signed on the users local device and protected by storing them only on internal storage and being encrypted while sent across the network. PuRSCA is highly scalable and is capable of handling large numbers of contracts in parallel. Also, it detects network errors and prevents the lose of funds by storing contracts pre-emptively, when the network fails during contract creation transactions.

Evaluations performed indicate that cost estimations proposed in this paper are accurate and including the deployment and transactions of SC costs with a high precision. In turn, these equations indicate as well as a guideline on how to implement SCs with least possible costs.

Finally, more inspections with respect to security aspects of PuRSCA need to be taken into consideration. The functionality of PuRSCA can be improved by adopting new SCs for setting up multi peer contracts. A platform for providing a supply and demand can be added to the front end of this system as a reference point for trading, such that information of available goods can be seen. Of course the implementation of such platform goes beyond PuRSCA and requires a carefully designed approach to maintain seller and buyers privacy and their private keys.

References

1. Airbnb. https://www.airbnb.com, Last visit September 28, 2017.
2. Android, Security Tipps; https://developer.android.com/training/articles/securitytips.html, Last visit July 31, 2017.
3. P. Arana: *Benefits and Vulnerabilities of Wi-Fi Protected Access 2 (WPA2);* http://cs.gmu.edu/~yhwang1/INFS612/Sample_Projects/Fall_06_GPN_6_Final_Report.pdf, Last visit July 30, 2017.
4. Bitcoin, Proof of Work; https://en.bitcoin.it/wiki/Proof_of_work, Last visit May 9, 2017.
5. A. Bogner, M. Chanson, A. Meeuw: *A Decentralised Sharing App Running a Smart Contract on the Ethereum Blockchain;* 6th International Conference on the Internet of Things(IoT16), Stuttgart, Germany, November 2016.
6. Brooklyn Microgrid; http://microgridmedia.com/brooklyn-startup-broadens-solar-power-access-with-p2p-energy-exchange/, Last visit May 9, 2017.
7. E. Bucher: *Zustandekommen des Vertrages*; http://www.eugenbucher.ch/pdf_files/Bucher_ORAT_10.pdf, Last visit May 9, 2017.
8. E. Bucher: *Kaufvertrag im Allgemeinen*. http://www.eugenbucher.ch/pdf_files/Bucher_ORBT_03.pdf, Last visit May 9, 2017.
9. D. Camps-Murr, A. Garcia-Saavedra, P. Serrano: *Device to Device Communications with WiFi Direct: Overview and Experimentation*. http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=81D8D0CEA0130A332E2EB14EC8563A1E?doi=10.1.1.725.7590&rep=rep1&type=pdf, Last visit July 20, 2017.
10. CryptoCompare.com: https://www.cryptocompare.com/, Last visit May 9, 2017.
11. Digix White paper; https://dgx.io/whitepaper.pdf, Last visit May 9, 2017.
12. Ebay. http://www.ebay.com, Last visit September 28, 2017.
13. Ebay, General Terms of Agreement; http://pages.ebay.ch/help/policies/user-agreement.html, Last visit May 9, 2017.
14. Ethereum, Documentation. http://www.ethdocs.org/en/latest/, Last visit May 9, 2017.
15. Ethereum.io: *Safe Remote Purchase;* http://solidity.readthedocs.io/, Last visit September 19, 2017.
16. Ethereum, Average Exchange Rate Since May 1, 2017; https://poloniex.com/, Last visit July 28, 2017.
17. Ethgasstation: https://ethgasstation.info, Last visit July 16, 2017.
18. Go-Ethereum: https://github.com/ethereum/go-ethereum, Last visit May 9, 2017.

19. K. Haataja, K. Hyppönen, S. Pasanen, P. Toivanen: Bluetooth Security Attacks, "Overview of Bluetooth Security", *SpringerBriefs in Computer Science*, 2013. http://www.springer.com/cda/content/document/cda_downloaddocument/9783642406454-c2.pdf?SGWID=0-0-45-1434420-p175453762, Last visit August 12, 2017.
20. F. Hawlitschek, T. Teubner, G. Henner: Understanding the Sharing Economy – Drivers and Impediments for Participation in Peer-to-Peer Rental, *49th Hawaii International Conference on System Sciences (HICSS)*, Koloa, HI, USA, Januar 2016, ISBN: 978-0-7695-5670-3.
21. IEEE 802.11 Wi-Fi Standards; http://www.radio-electronics.com/info/wireless/wi-fi/ieee-802-11-standards-tutorial.php, Last visit July 21, 2017.
22. QrGen; https://github.com/kenglxn/QRGen, Last visit May 9, 2017.
23. Skip The Rental Counter, https://turo.com/, Last visit August 22, 2018.
24. Suisseld Multi Signing Platform; https://www.multisigning.ch/, Last visit August 4, 2017.
25. Swiss Federal Council: *Federal Law on the Supplement to the Swiss Civil Code;* https://www.admin.ch/opc/de/classified-compilation/19110009/index.html#a1, Last visit Sep 28, 2017.
26. N. Szabo: The Idea of Smart Contracts [1997]; http://www.fon.hum.uva.nl/, Last visit May 9, 2017.
27. Transactive Grid; https://www.slideshare.net/JohnLilic/transactive-grid, Last visit May 9, 2017.
28. S. Viehböck: Brute Forcing Wi-Fi Protected Setup, https://sviehb.files.wordpress.com/2011/12/viehboeck_wps.pdf, Last visit July 11, 2017.
29. Web3j library; https://github.com/web3j/web3j, Last visit September 22, 2017.
30. S. Rafati, F. Schüpfer, T. Bocek, and B. Stiller: A Peer-to-Peer Purchase and Rental Smart Contract-based Application, IfI Technical Report No. 2017-04, Department of Informatics, University of Zürich, Switzerland, August 2017, https://files.ifi.uzh.ch/CSG/staff/Rafati/Purchase-Rental-APP-SC.pdf.
31. F. Schüpfer: *Design and Implementation of a Smart Contract Application*, Master Thesis, Communication Systems Group, Department of Informatics, University of Zürich, Switzerland, August 2017, https://files.ifi.uzh.ch/CSG/staff/Rafati/Florian-Schupfer-MA.pdf.
32. J. Burger: *Collaborative DDoS Mitigation based on Blockchains*, Bachelor Thesis, Communication Systems Group, Department of Informatics, University of Zürich, Switzerland, August 2017, https://files.ifi.uzh.ch/CSG/staff/Rafati/Jonathan-Burger-BA.pdf.

Bionotes

Sina Rafati Niya
Communication Systems Group CSG, Department of Informatics IfI, University of Zürich, Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
rafati@ifi.uzh.ch

Sina Rafati Niya is with the Communications Systems Group CSG, Department of Informatics IfI, University of Zurich UZH, started his Ph. D. in 2016 under the supervision of Prof. Dr. Burkhard Stiller. He holds a Master's degree in Information Technology engineering from Urmia University of Technology Iran, In 2015. Since 2016 he has been involved in developing blockchain-based Dapps and researching on the usability of Blockchains in the Internet-of-Things (IoT) domain.

Florian Schüpfer
University of Zürich, Department of Informatics IfI, Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
flo.schuepfer@bluewin.ch

Florian Schüpfer is a Master student at the University of Zurich, Switzerland. He did his Master thesis with the CSG in the blockchain area with the main focus on P2P trading applications.

Prof. Dr. Thomas Bocek
Communication Systems Group CSG, Department of Informatics IfI, University of Zürich, Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
bocek@ifi.uzh.ch

Prof. Dr. Thomas Bocek was the head of P2P and Distributed Systems at the Communication Systems Group CSG, Department of

Informatics IfI, University of Zurich from 2013 to 2018. Since then he holds a professor's position with the Hochschule für Technik Rapperswil. Before that, Thomas worked as a software engineer and technical project manager in the financial sector. Thomas is mainly interested in communication systems and networks, especially focusing on P2P, distributed systems, and blockchains including Bitcoin and Ethereum. He is also involved in the blockchain start-up modum.io, which combines IoT sensor devices with blockchains.

Burkhard Stiller
Communication Systems Group CSG, Department of Informatics IfI, University of Zürich, Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
stiller@ifi.uzh.ch

Prof. Dr. Burkhard Stiller received the Diplom-Informatiker (M.Sc.) degree in Computer Science and the Dr. rer.-nat. (Ph. D.) degree from the University of Karlsruhe, Germany. He has been a Full Professor of the Communication Systems Group, Department of Informatics, University of Zurich since 2004. He held previous research positions with the Computer Laboratory, University of Cambridge, U.K., the Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland, and the University of Federal Armed Forces, Munich, Germany. He did coordinate various Swiss and European industrial and research projects, such as BC4CC, Foodchains, AA-MAIS, DAMMO, SmoothIT, SmartenIT, SESERV, and Econ@Tel, besides participating in others, such as M3I, Akogrimo, EC-GIN, EMANICS, FLAMINGO, symbIoTe, and ACROSS. His main interests are published in well over 250 research papers and include systems with a fully decentralized control (blockchains, clouds, peer-to-peer), network and service management (economic management), Internet-of-Things (security of constrained devices, LoRa), and telecommunication economics (charging and accounting). He is also involved as a senior advisor in the blockchain start-up modum.io, which combines IoT sensor devices with blockchains.